VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Optimální řešení Lazy evaluation pro LZ77

Optimal Solution of Lazy Evaluation for LZ77 based Encoder

2012 Eva Čecháková

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Eva Čecháková**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Optimální řešení Lazy evaluation pro LZ77
Optimal Solution of Lazy Evaluation for LZ77 based Encoder

Zásady pro vypracování:

Cílem práce je navrhnout algoritmus,který by úspěšně řešil problém vhodného zvolení kódových párů sekvence v algoritmu LZ77 při použití Lazy Evaluation.
Úkolem bude zjistit, zda lze za cenu zvýšení výpočetní náročnosti zlepšit dosažený kompresní poměr při použití obecného Lazy Evaluation principu.

Obsah práce:
1. Popis metody LZ77 a jejich variant a principu Lazy Evaluation.
2. Popis problému s volbou vhodné kombinace sekvencí generovaných při Lazy Evaluation.
3. Návrh metod pro řešení uvedeného problému.
4. Implementace metod.
5 Testovaní efektivity metod.

Seznam doporučené odborné literatury:

Data Compression: The Complete Reference, David Salomon, 4. ed., Springer, 2007

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Jan Platoš, Ph.D.**

Datum zadání: 18.11.2011
Datum odevzdání: 04.05.2012

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracovala samostatně. Uvedla jsem všechny literární prameny a publikace, ze kterých jsem čerpala.

V Řepištích 22. dubna 2012 .....

Poděkování

Ráda bych tímto poděkovala panu Ing. Janu Platošovi, Ph.D. za odborné vedení při vytváření mé bakalářské práce a mé rodině za podporu a trpělivost.

Abstrakt

Cílem této bakalářské práce je bližší seznámení s kompresní metodou LZ77, jenž se řadí mezi adaptivní slovníkové metody bezztrátové komprese textu, její variantou LZSS a s principem opožděného vyhodnocování. Následuje využití získaných znalostí při rozboru problému nejlepšího výstupního kódování a návrhu jeho řešení. Výsledkem této práce je implementace optimalizovaných metod LZ77 a LZSS v jazyce C#. Součástí práce je také testování nově vytvořených algoritmů na pěti textových souborech z Canterbury korpusu a zhodnocení jejich efektivity, tedy k jakému došlo zlepšení kompresního poměru oproti původním metodám LZ77 a LZSS.

Klíčová slova

komprese dat, bezztrátová komprese, komprese textu, LZ77, LZSS, opožděné vyhodnocování

Abstract

The aim of this thesis is closer acquaintance with LZ77 compression method, which is one of adaptive dictionary methods of lossless text compression. Its variant LZSS and the principle of lazy evaluation are covered. The acquired knowledge is used in the analysis of the best output encoding problem, and design of a solution. The result of this thesis is implementation of the optimized methods LZ77 and LZSS in C #. The thesis also includes testing of newly developed algorithms using five text files from the Canterbury corpus and evaluating their effectiveness, in other words, the improvement in compression ratio compared to the original methods LZ77 and LZSS.

Key words

data compression, lossless compression, text compression, LZ77, LZSS, lazy evaluation

Obsah

Seznam tabulek

Seznam obrázků

1 Úvod

Již odedávna mají lidé touhu shromažďovat různé informace. Avšak v současné době je tento trend silnější než kdy dříve. Proto je potřeba neustále přemýšlet nad tím, jak tyto informace neboli v elektronické podobě též nazývána data co možná nejefektivněji uchovávat. V dnešním světě rychlého vývoje informačních technologií se to jeví jako velmi snadný úkol. Opak je pravdou. Nemohu sice popírat, že za posledních několik let především u datových nosičů a přenosových médií, jako jsou například pevné disky, flash disky, paměťové karty, metalické a optické kabely, došlo k výraznému zvýšení kapacity. Přesto oproti nárůstu velikostí a především počtu datových souborů se jedná spíše jen o mírný růst.

Již dříve pochopili lidé jako Shannon a Huffman, že je třeba vyvíjet a zlepšovat metody, které by zmenšovaly velikost ukládaných nebo přenášených dat (datových souborů). Těmto metodám se obecně říká komprese dat. Širší laická veřejnost zná toto sousloví především ve spojení se zvukovými, obrazovými a video soubory. Dalším typem datových souborů, u nichž je třeba používat kompresi, jsou soubory textové. Setkáváme se s nimi mnohem častěji, než si uvědomujeme – publikace v elektronické podobě, e-maily, zálohy databází, nejrůznější dokumentace a formuláře.

Mým cílem v této bakalářské práci je podrobněji nastudovat kompresní metodu LZ77 a její varianty, jenž se řadí mezi adaptivní slovníkové metody bezztrátové komprese textu. Získané poznatky dále zhodnotit při vytváření jakési optimalizované metody LZ77 a LZSS, která by měla být schopna dosáhnout lepších výsledku než původní metoda.

Obsah práce je rozdělen do dvou hlavních částí, teoretická, kde se řadí kapitoly 1 – 5 a praktická, kapitoly 6 a 7. První a druhá kapitola je obecným úvodem do problematiky komprese dat. Ve třetí a čtvrté kapitole popisuji slovníkovou kompresní metodu LZ77, její variantu LZSS, principy obou metod a řeším ukázkové příklady. Pátá kapitola rozebírá opožděné vyhodnocování a vliv jeho použití na LZ77 a LZSS. V šesté kapitole je popsán rozbor a řešení problému s nejlepším kódováním. Sedmá kapitola se věnuje testování navržené optimalizace LZ77 a LZSS pomocí pětice vybraných souborů z Canterbury korpusu. A poslední osmá kapitola je zhodnocením celé práce, především pak výsledků testů ověřujících správnost získaných poznatků.

2 Komprese dat

Komprese (neboli komprimace či zhušťování dat) je speciální druh kódování dat za účelem zmenšení jejich objemu odstraněním nadbytečné informace (redundance).[6] Toto kódování je určeno daným kompresním algoritmem, pomocí kterého jsme schopni jasně definovat postup komprese i zpětný postup dekomprese pro rekonstrukci původních dat. Kompresní algoritmus převádí **zdrojové jednotky** (symboly a posloupnosti symbolů – takzvaná slova a posloupnosti slov) na **posloupnosti bitů**.[6] Zdrojové jednotky se mohou označovat jako **vzory** (originály) a výsledné posloupnosti jako **obrazy**.[6]

Důvodů proč komprimovat data a tím zmenšovat jejich velikost je mnoho, například pro archivaci souborů, přenos těchto souborů přes síť s omezenou rychlostí nebo kvůli omezené datové propustnosti.

Na vyjádření efektivnosti metod komprimace můžeme použít **komprimační (kompresní) poměr**, který je definován následovně:

$$komprimační\ poměr = \frac{\text{velikost obrazu}}{\text{velikost vzoru}}$$

kde například hodnota 0,8 znamená, že údaje pro komprimaci zabírají 80% původního paměťového prostoru. Hodnoty větší než jedna znamenají negativní, obvykle nežádoucí expanzi zpracovaných dat.[5]

2.1 Druhy komprese

Základní vlastností rozdělující kompresní metody na dvě velké skupiny je **ztrátovost**. Ta určuje, zda při procesu komprese došlo k menší ztrátě dat či nikoliv.

2.1.1 Ztrátová

Ztrátová komprese je založena na odstraňování nepotřebných informací (detailů). Používá se hlavně pro kompresi zvukových, obrazových a video souborů, kde využívá nedokonalosti lidského zraku a sluchu. V praxi to znamená, že například u zvukových souborů se odstraňují signály mimo slyšitelný frekvenční rozsah nebo slabší signály, které jsou překrývány silnějšími. U obrazových souborů je možné kompresí snížit barevnou hloubku nebo rozlišení podle potřeby. Pro video soubory se často používají kombinace předešlých kompresních metod pro obraz a zvuk. Patří zde například: MPEG, JPEG, H.264.

2.1.2 Bezztrátová

Základní vlastností bezztrátové komprese je totožnost původních dat s daty dekomprimovanými – vždy zachovává kompletní informaci. Využívá se především u textových a binárních souborů nebo u vektorové grafiky.

Dle přístupu ke kompresi můžeme tuto skupinu dále dělit na:

- **Statické algoritmy** – stanovuje se pravděpodobnost pro každý symbol ze vstupu a podle toho se vytváří co nejoptimálnější kód.[2]

- **Slovníkové algoritmy** – nahrazuje často se vyskytující posloupnosti znaků, které ukládá do slovníku, za odpovídající kódy (kódová slova).
    - **Statické** – během procesu komprese se slovník nemění.
    - **Semiadaptivní** – slovník se během komprese mění (vytváří) podle aktuálně komprimovaných dat a je jejich součásti pro pozdější dekompresi.
    - **Adaptivní** – stejný postup jako u semiadaptivních algoritmů, pouze slovník není součástí komprimovaných dat, protože při dekomprimaci lze tento slovník opětovně vytvořit. Zde řadíme algoritmus **LZ77** a jeho varianty, který budu podrobněji popisovat v kapitole 3.

2.1.3 Další rozdělení

Dle výpočetní náročnosti:

- Symetrické
- Asymetrické

Dle zpracování vstupních dat:

- Proudové
- Blokové

Dle počtu průchodů:

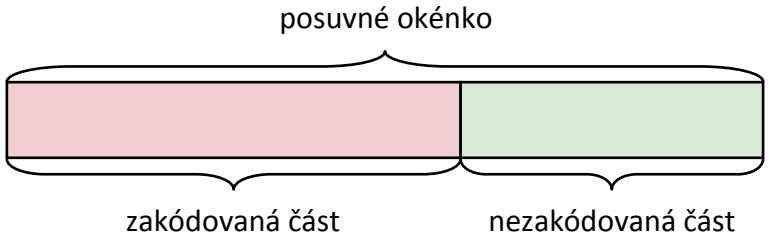
- Jedno-průchodové
- Více-průchodové

3 LZ77

Bezztrátový kompresní algoritmus LZ77, někdy také označován jako LZ1,[1] byl pojmenován podle Abrahama Lempla a Jacoba Ziva, kteří ho publikovali v roce 1977. Řadí se mezi slovníkové algoritmy a v současnosti je jedním z nejpoužívanějších kompresních algoritmů.

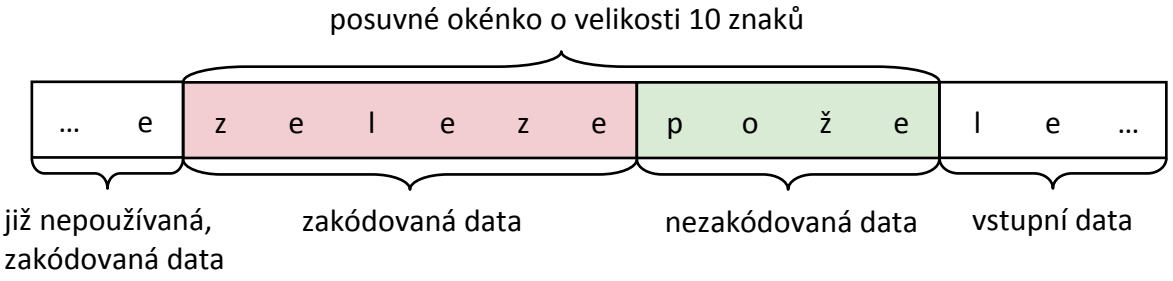
3.1 Princip

Algoritmus LZ77 využívá takzvaného posuvného okénka neboli anglicky sliding window. Při komprimaci se nezakódovanými daty (textem) posouvá pomyslné okénko. Toto okénko se skládá ze dvou částí, zakódovaná (anglicky search buffer) a nezakódovaná část (anglicky look-ahead buffer), znázorněno na obrázku 1.



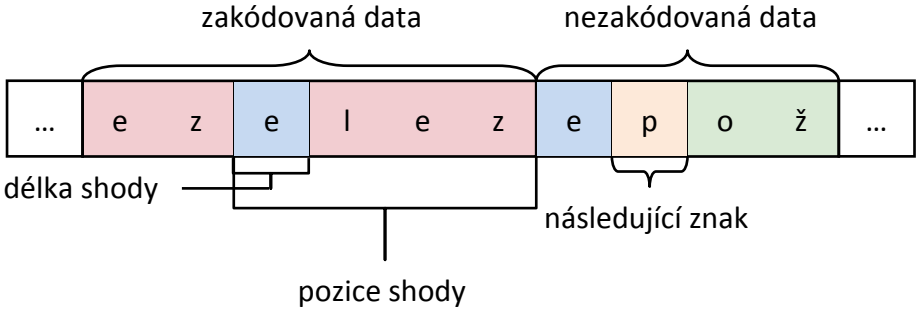
Obrázek 1 - Posuvné okénko

Z obrázku je patrné, že pro popis okénka jsou důležité dva parametry, velikost okénka a velikost nezakódované části. Velikost okénka je různá, většinou se setkáme s velikostí od 2048 bytů až po několik MB. U nezakódované části se velikost pohybuje v několikanásobně menším rozmezí, mezi 8 až 256 znaky.[2] Algoritmus pracuje pouze s daty v okénku, a tak paměťová náročnost algoritmu u různé délky vstupního řetězce zůstává konstantní, jak lze vidět na obrázku 2.



Obrázek 2 – Zaplněné posuvné okénko

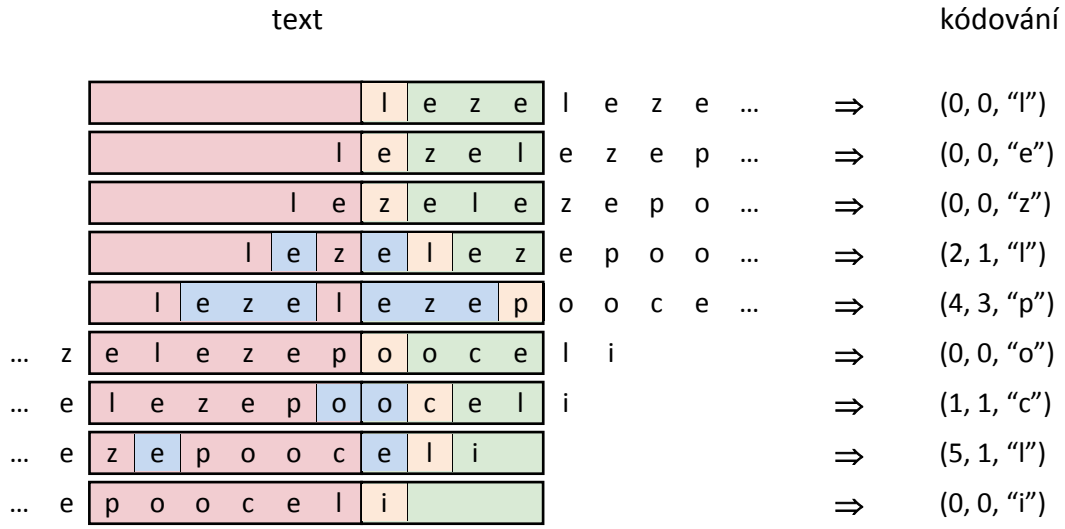
Samotný princip algoritmu spočívá v prohledávání zakódované části směrem zprava doleva a nalezení co nejdelší shody pro sekvenci znaků z nezakódované části. Výsledkem je **trojice** parametrů, která je odesílána na výstup – **(pozice, délka, následující znak)**, kde **pozice** je vzdálenost nalezené shody (shodných znaků) od konce zakódované části, **délka** vyjadřuje počet znaků nalezené shody a **následující znak** je prvním znakem v nezakódované části, který již není obsažen ve znacích, pro které byla nalezena shoda, viz obrázek 3, kde výsledná trojice bude vypadat takto – (4, 1, "p").



Obrázek 3 - Kódová trojice

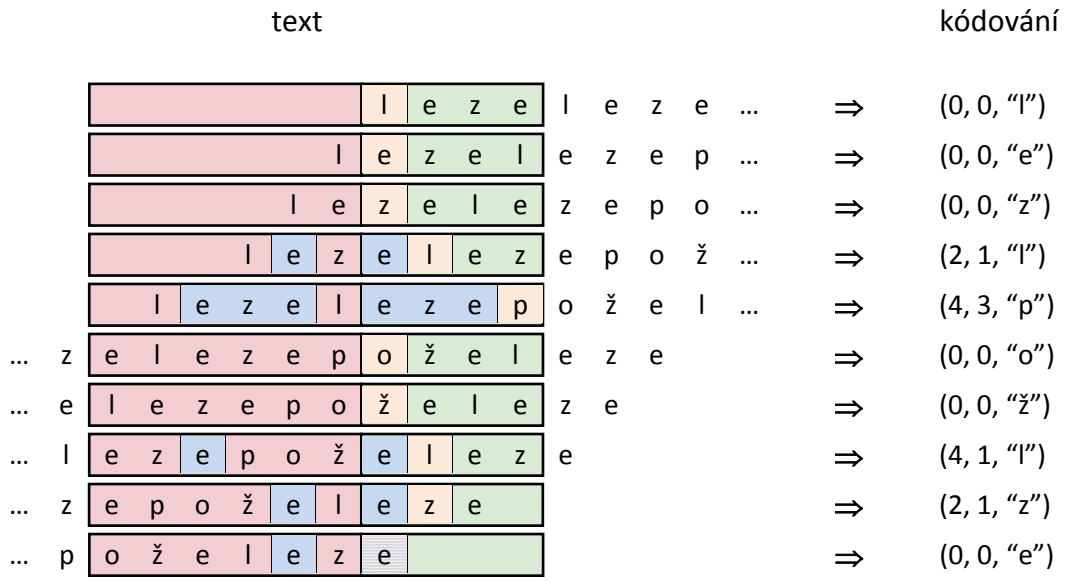
Důvodem pro uvádění třetího parametru (následující znak) v trojici je situace, kdy pro daný znak z nezakódované části nebyl nalezen žádný shodný znak v zakódované části. V tomto případě se pozice i délka nastaví na hodnotu nula a třetí parametr – následující znak je nastaven na tento znak – (0, 0, znak).[4]

Při provádění komprimace mohou nastat dva případy. V prvním případě se na konci textu vyskytuje znak, pro který neexistuje shoda, jak lze vidět na obrázku 4.



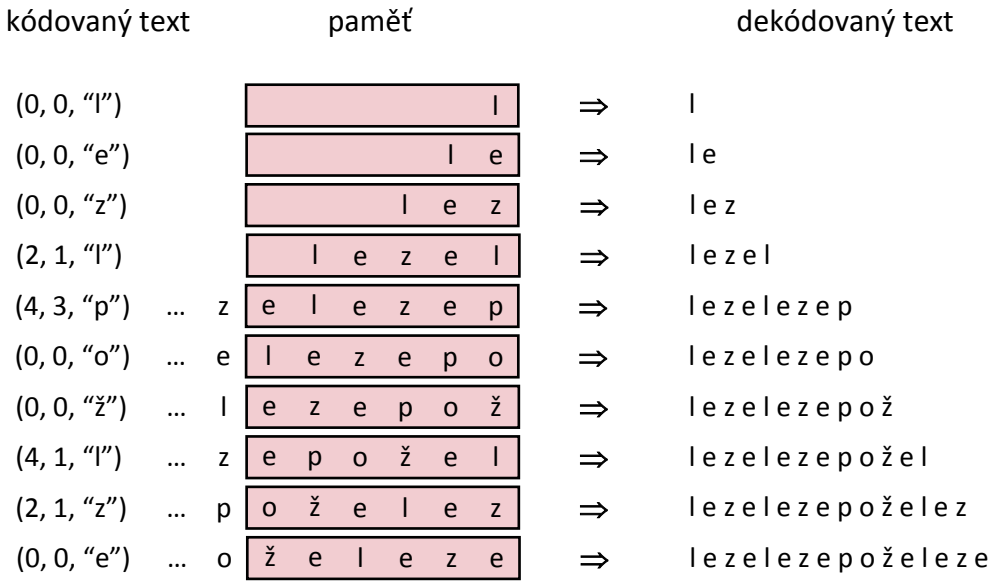
Obrázek 4 - Komprese LZ77 na konci bez shody („*lezelezepooceli*")

Ve druhém případě se na konci textu vyskytuje znak, který je obsažen v sekvenci shodných znaků nebo se jedná o shodu jediného znaku. Délku shody zmenšíme o jedna a parametr následující znak nabude hodnoty posledního znaku shody. Postup je znázorněn na obrázku 5.



Obrázek 5 - Komprese LZ77 na konci se shodou (*„lezelezepoželeze“*)

Za zmínku jistě stojí fakt, že dekomprese u algoritmu LZ77 je velmi jednoduchá a daleko méně náročná než komprese, viz obrázek 6. Dekompresní program si v paměti udržuje dekomprimovaný text v délce zakódované části. Čte trojice a z nich dekóduje opakující se části textu a znaky za nimi. Na dekompresi je podstatné, že je velmi rychlá. Zatímco komprese pro nalezení shody vyžaduje prohledávání zakódované části, dekomprese shodu pomocí pozice zjistí okamžitě.[3]



Obrázek 6 – Dekomprese LZ77 („*lezelezepoželeze*“)

4 LZSS

Mluvíme-li o algoritmu LZSS, jedná se v podstatě o efektivnější variantu kompresní metody LZ77. Řadí se mezi nejpoužívanější z modifikací algoritmu LZ77. Implementace této metody byla navržena Bellem v roce 1986 podle idejí publikovaných J. Storerem a T. Szymanskim v roce 1982.

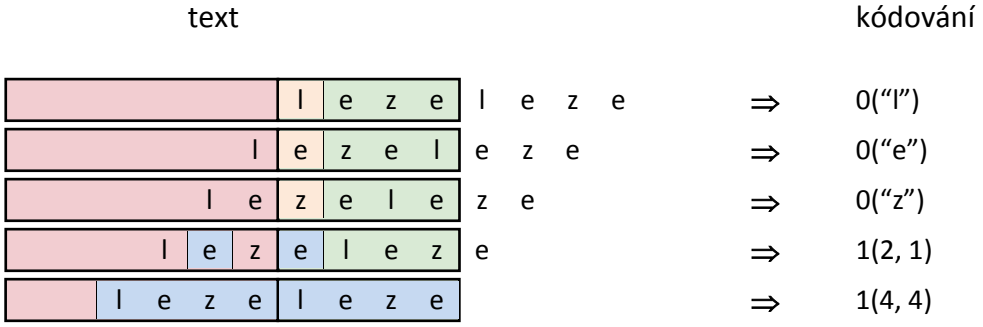
4.1 Princip

Jednou z největších nevýhod algoritmu LZ77 je vytváření kódových trojic za každých okolností. V případě, že pro aktuálně kódovaný znak nebyla nalezena žádná shoda (v nezakódované části okénka se tento znak nenachází) nebo byla nalezena shoda pouze pro tento jediný znak, je na výstup poslána opět trojice. Velikost celé trojice je mnohem větší než velikost samotného znaku, který tato trojice popisuje.

Pro výše zmiňovaný problém byla navržena právě metoda LZSS, jejíž výstupní data jsou tvořena **dvojicemi** parametrů nebo samostatnými **znaky**. Dvojice má tento tvar – **(pozice, délka)**, kde opět **pozice** vyjadřuje vzdálenost nalezené shody (shodných znaků) od konce zakódované části a **délka** vyjadřuje počet znaků nalezené shody. Obě čísla se ukládají jako čistý bitový zápis a tedy pro uložení je **pozice** je potřeba $\log_2 N$ bitů, kde N je velikost okénka, a pro kódování **délky** je potřeba $\log_2 M$ bitů, kde M je maximální délka shody.[2] Znak ukládáme přímo jako osmibitové číslo.[2] Díky těmto poznatkům můžeme jednoduše určit, jestli je vhodné uložit dvojici nebo samostatný znak, a to pomocí vzorce

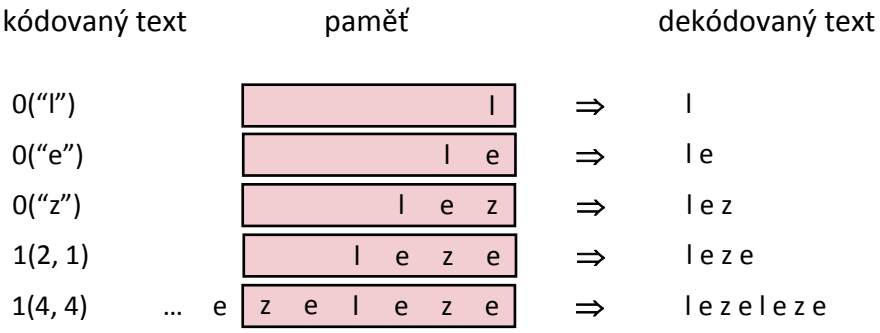
$$d = \left\lceil \frac{\log_2 N + \log_2 M}{8} \right\rceil,$$

kde **d** představuje minimální délku shody, pro kterou už je výhodné použít kódovou dvojici. Celý postup je znázorněn na obrázku 7, příklad je uveden pro tyto hodnoty: N = 6, M = 4, d = 1.



Obrázek 7 – Komprese LZSS (*„lezeleze“)*

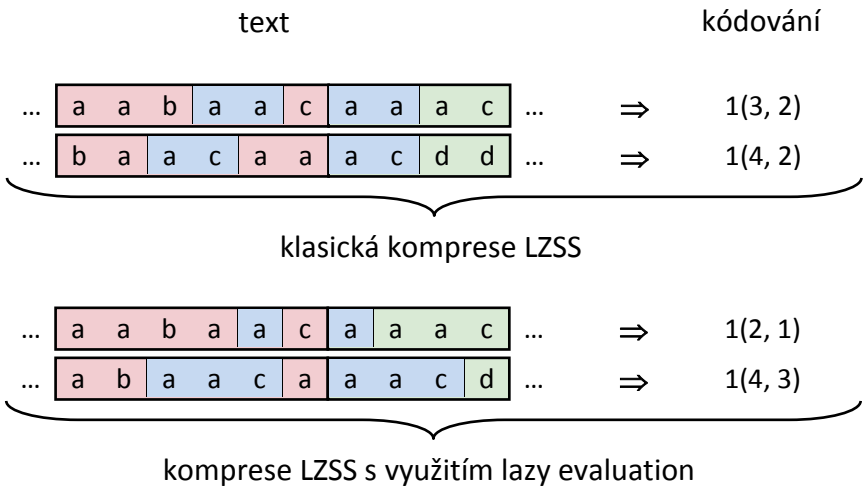
Takzvaný pomocný bit, který je přidávám před dvojici nebo znak, slouží při dekompresi k určení toho, zda bude zpracovávána dvojice nebo znak. Jednotlivé kroky dekomprese lze vidět na obrázku 8.



Obrázek 8 - Dekomprese LZSS („*lezeleze*“)

5 Lazy evaluation

Opožděné vyhodnocování (anglicky lazy evaluation či lazy matching) je metoda, kterou se dá dosáhnout velkého vylepšení kompresního poměru.[2] Z jiného hlediska je opožděné vyhodnocování programovací technikou, která odkládá vyhodnocení výrazu až na chvíli, kdy je toho zapotřebí. Tohoto tvrzení lze využít i u LZ77 a LZSS, a to tak, že kompresní algoritmus nezakóduje ihned první nalezenou shodu na pozici *p*, ale vyzkouší, zda na pozici *p* + 1 není nalezena delší shoda. V případě, že delší shoda nalezena není, pokračuje algoritmus obvyklým způsobem. V případě, že delší shoda nalezena je, postupuje algoritmus tak, že znak na pozici *p* je zakódován samostatně, shoda na pozici *p* + 1 zatím kódovaná není a opět se zkouší shoda na další pozici, tentokrát *p* + 2. Tento cyklus se opakuje, dokud nedosáhne předem určeného limitu – maximálního počtu pokusů. Vše lépe vysvětlí obrázek 9.



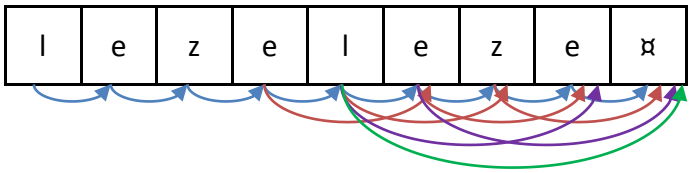
Obrázek 9 – LZSS s a bez lazy evaluation

6 Řešení problému nejlepšího kódování

I s pomocí lazy evaluation, které výrazně zlepší kompresní poměr, stále existují případy, kdy na výstupu nezískáme nejlepší kódování. V prvním případě mějme nalezené shody o délce 2, 3, 5 a 4 přesně v tomto pořadí. Pro každou ze shod délky 2 a 3 se zakóduje jeden samostatný znak, poté se zakóduje shoda délky 5. Na výstupu máme tři trojice (LZ77). Mnohem výhodnější by ale bylo namísto dvou samostatných znaků zakódovat jedinou shodu o délce 2 a poté shodu délky 5. Na výstupu máme rázem dvě trojice oproti původním třem. V druhém případě mějme shody 3, 1, 5, 1 a 1. Nejprve se zakóduje shoda délky 3 a pak dva samostatné znaky. Pro tuto sekvenci shody by nejlepší kódování znamenalo, že se zakóduje shoda délky 2 a pak 5, výsledkem jsou jen dvě trojice.

Oba předchozí případy jsou hlavními překážkami na cestě k výraznému zmenšení velikosti zkomprimovaného souboru. Řešení není nijak složité. Celý postup je možné shrnout do několika bodů:

1) Vyhledáme nejlepší shodu pro každý jednotlivý znak vstupního textu a tyto data uložíme do vhodné datové struktury, například pole. Nesmíme opomenout, že shodu délky 2 můžeme rozložit na dvě shody o délce 1, viz obrázek 10.

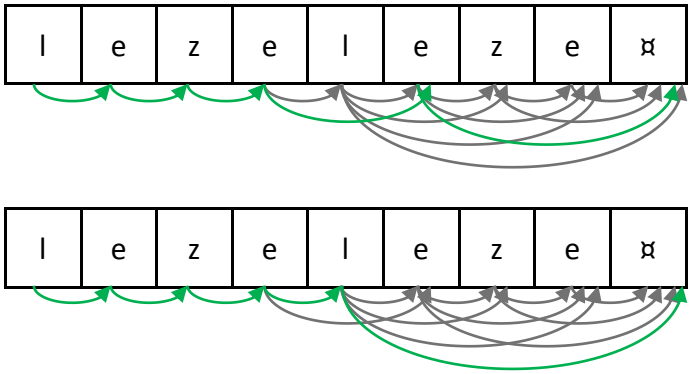


Obrázek 10 – Všechny možné kombinace shod (¤ - pomocný koncový znak)

   - příklad: „*lezeleze*“ (velikost okénka = 256, maximální délka shody = 16, algoritmus LZ77)
     (0, 0, “l”); (0, 0, “e”); (0, 0, “z”); (2, 1, “l”); (4, 3, “e”); (4, 2, “e”); (4, 1, “e”); (0, 0, “e”)

2) Dále zpětně projdeme toho pole (zezadu dopředu) a sečteme jednotlivé kroky (počty bitů) od posledního prvku až k tomu aktuálnímu pro veškeré kombinace shod a tu nejlepší (s nejmenším počet bitů) uložíme do pomocné proměnné přiřazené aktuálnímu prvku. Projdeme-li pole pomocí výsledných hodnot z pomocných proměnných, získáme nejlepší kódování, znázorněno na obrázku 11.
   - (0, 0, “e”) – krok 1
     (4, 1, “e”) – krok 1; (0, 0, “z”) – krok 2
     (4, 2, “e”) – krok 1; (4, 1, “z”) – krok 2; (0, 0, “e”) – krok 3
     (4, 3, “e”) – krok 1; (4, 2, “z”) – krok 2; (4, 1, “e”) – krok 3; (0, 0, “l”) – krok 4
     (2, 1, “l”) – krok 2; (0, 0, “e”) – krok 3

(0, 0, “z”) – krok 3
(0, 0, “e”) – krok 4
(0, 0, “l”) – krok 5
=> pro zakódování textu „*lezeleze*“ je potřeba minimálně 5 trojic



Obrázek 11 – Nejlepší kombinace shod (¤ - pomocný koncový znak) – nahoře zobrazen postup při klasické LZ77, dole zobrazen postup s využitím opožděného vyhodnocování

7 Testování

Nyní přichází na řadu testování programu vytvořeného na základě získaných poznatků z předchozích kapitol. Účelem těchto testů je potvrdit zvýšení efektivity optimalizovaných kompresních algoritmů LZ77 a LZSS oproti původním a to pomocí vhodně zvolených textových souborů, které jsou součástí takzvaného Canterbury korpusu [7]. Jedná se o následujících pět souborů:

Soubor	Formát	Jazyk	Velikost (v bytech)	Počet tisknutelných znaků (vč. mezer)	Počet slov
aaa.txt	text	angličtina	100 000	100 00	1
alice29.txt	text	angličtina	152 089	26 690	144 873
bible.txt	text	angličtina	4 047 392	766 112	4 017 009
lcet10.txt	text	angličtina	426 754	411 716	62 786
world192.txt	text	angličtina	2 473 400	2 343 162	326 119

Tabulka 1 – Testovací soubory

Další proměnnou, která bude hrát při testování důležitou roli, bude zvolené kódování pro pozici a délku. Použila tyto jsem tři typy kódování:

- **Přímé kódování** – přímý bitový zápis hodnot do souboru[2]
- **Fibonacciho kódování** – kódování přirozených čísel kódem variabilní délky[2]
- **B-blokové kódování** – kódování přirozených čísel, kde velikost výstupního zápisu je závislá na parametru *b*, který představuje základ kódování[2]

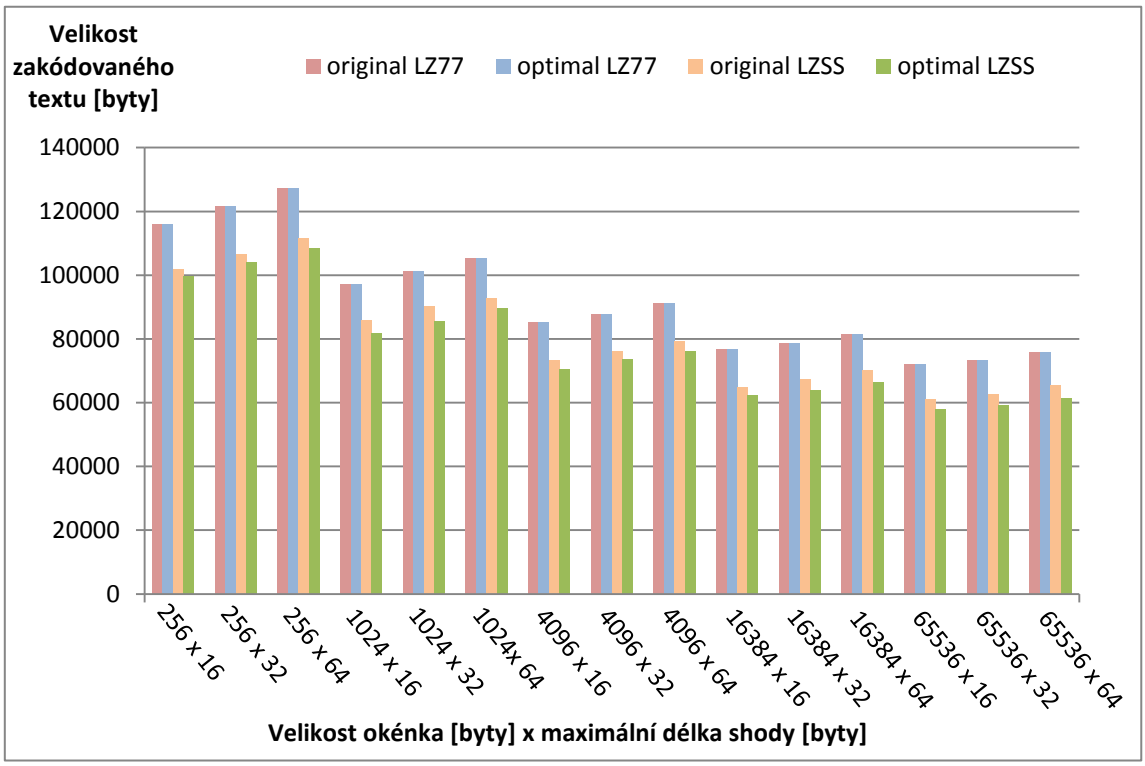
Pro znak je vždy určeno 8 bitů.

Předmětem testování je také volba ideální kombinace velikosti okénka a maximální délky shody v závislosti na použitém typu kódování. Touto problematikou se budu více zabývat v následujících testech.

7.1 Testování souborů alice29.txt

Na tomto souboru budu podrobně testovat různé typy kódování pro pozici a délku a jejich kombinace. U B-blokového kódování bude použita různá hodnota parametru *b*. Testování bude provedeno pro velikost okénka 256B, 1kB, 4kB, 16kB, 64kB a pro maximální délku shody 16B, 32B, 64B. Nejlepší výsledná nastavení pak budou použita i na další testovací soubory.

7.1.1 Přímé kódování pozice i délky

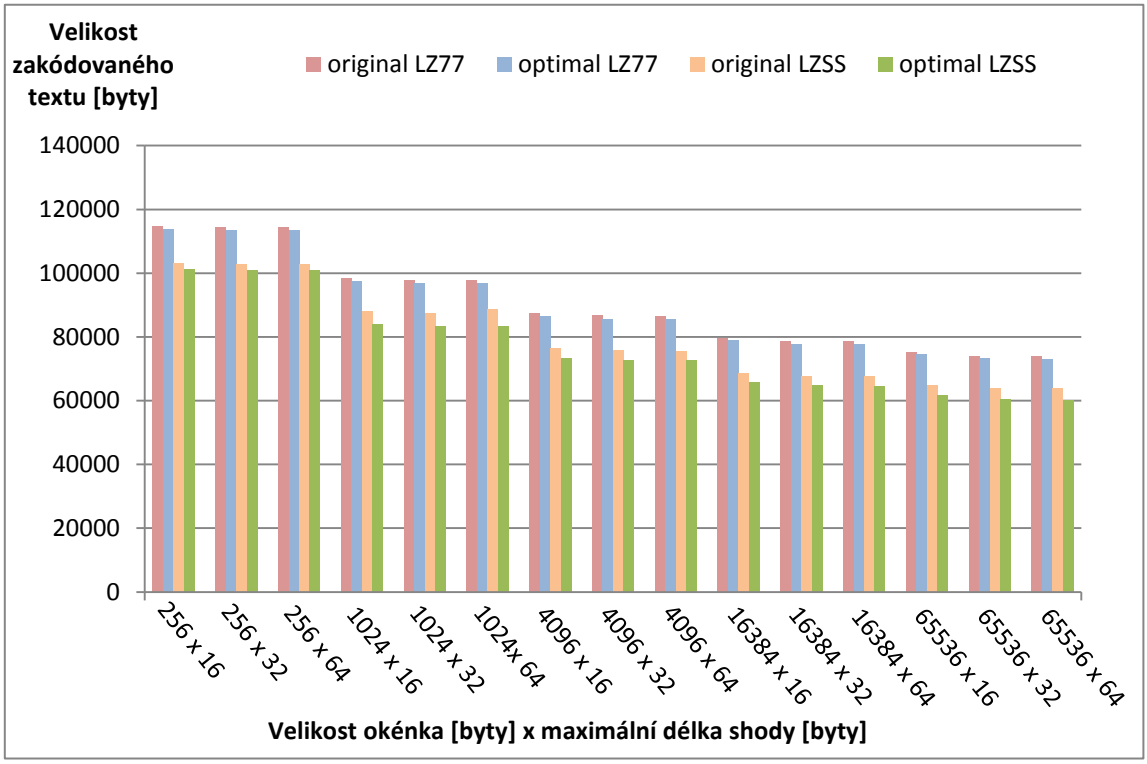


Obrázek 12 – Graf alice29.txt – přímé kódování (pozice, délka)

Z grafu na obrázku 12 je patrné, že pro přímé kódování obou parametrů se velikost zakódovaného textu snižuje společně s rostoucí velikosti okénka, tudíž nejlepších výsledků bylo dosaženo pro velikost 65 536B (64kB). Zároveň při použití přímého kódování na délku shody nastává případ, kdy pro každou nalezenou shodu (nezávisle na její délce) kódujeme maximální délku shody, tedy jednu ze zvolených variant 16B, 32B nebo 64B. Předchozí tvrzení potvrzuje graf, ve kterém se jako nejlepší volba maximální délky shody pro přímé kódování jeví délka 16B.

Dle očekávání se jako nejúčinnější kompresní algoritmus ukázal nově vytvořený optimal LZSS. U Optimal LZ77 nedošlo oproti original LZ77 k žádnému zlepšení, protože při přímém kódování a stejném počtu trojic k němu ani dojít nemůže.

7.1.2 Přímé kódování pozice, Fibonacciho kódování délky

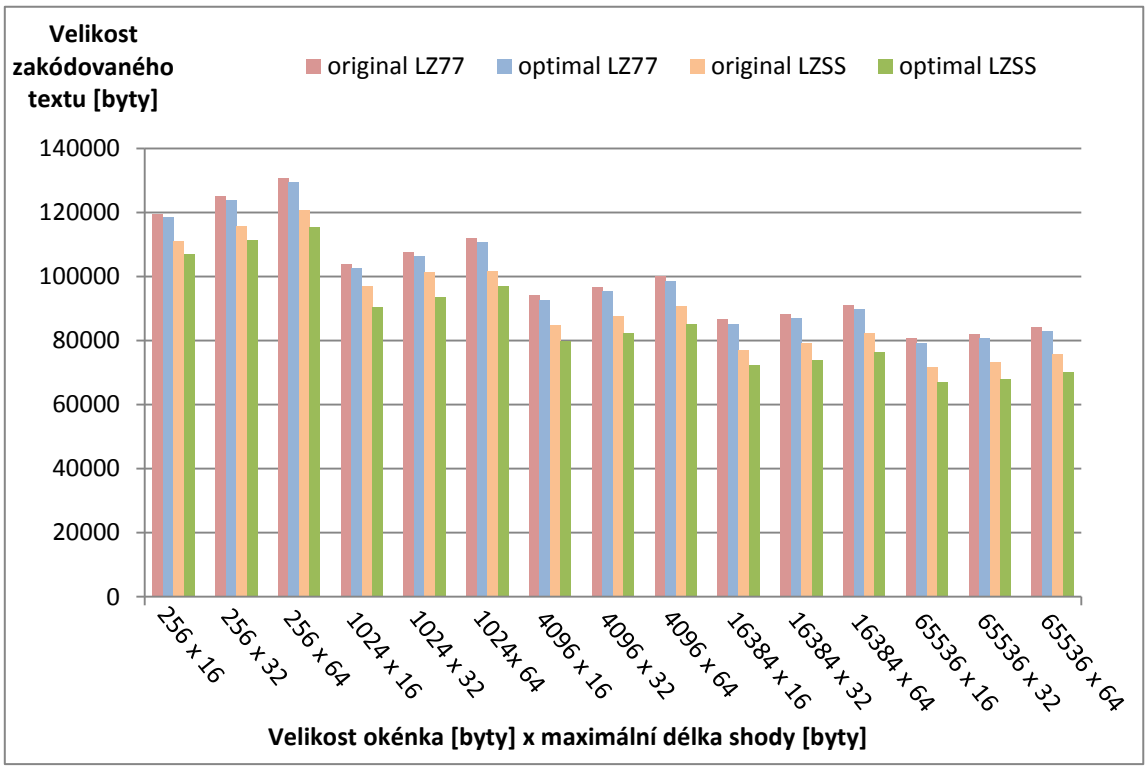


Obrázek 13 – Graf alice29.txt – přímé kódování (pozice), Fibonacciho kódovaní (délka)

Výsledky testu znázorněny na obrázku 13 opět vyhodnocují velikost okénka 65 536B (64kB) jako nejlepší pro tuto kombinaci přímého a Fibonacciho kódování. Právě díky použitému Fibonacciho kódování na délku shody, při kterém se kóduje konkrétní délka, je nevýhodnější zvolit 64B jako maximální délku shody. Avšak v porovnání s předchozím testem (7.1.1), kde bylo použito přímé kódování na pozici i délku, má tato varianta horší výsledný kompresní poměr, a to o 1,53% (porovnáno pro nejlepší výsledky algoritmu optimal LZSS obou testů).

Celkově pak došlo ke zlepšení efektivity u optimal LZ77 průměrně o necelých 900B oproti original LZ77, neúčinnějším algoritmem zůstává nadále optimal LZSS.

7.1.3 Fibonacciho kódování pozice, přímé kódování délky

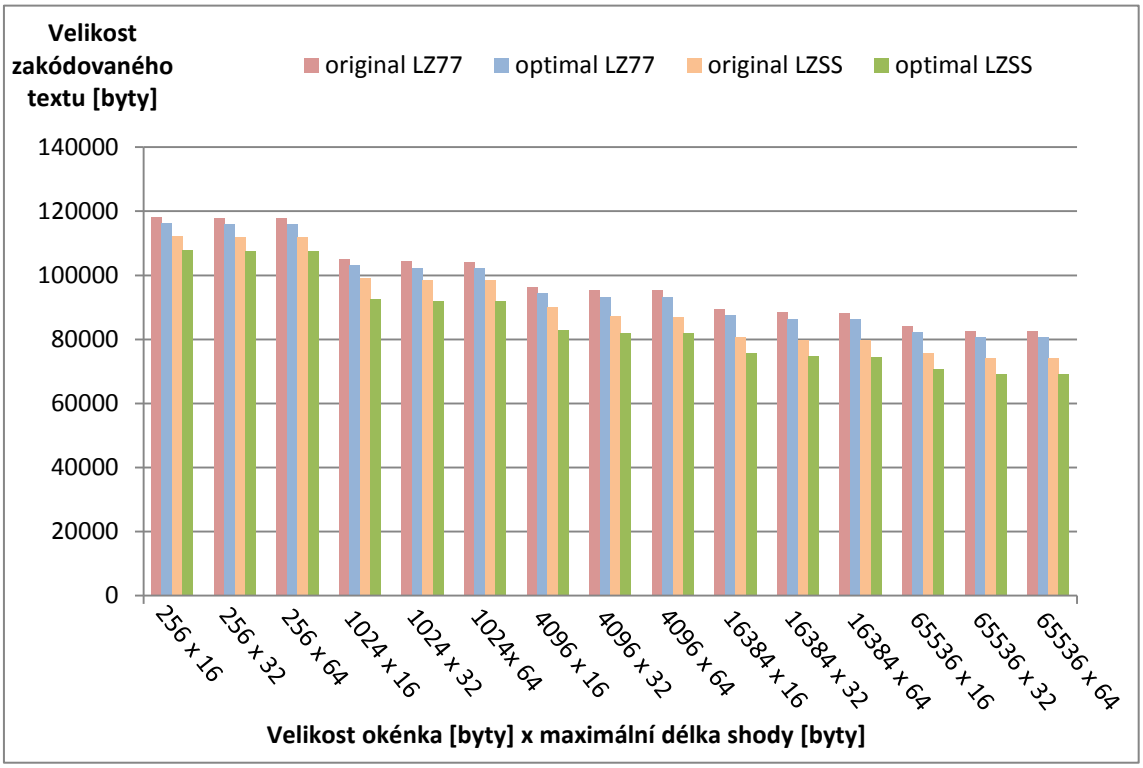


Obrázek 14 – Graf alice29.txt – Fibonacciho kódování (pozice), přímé kódovaní (délka)

Z výše uvedeného grafu (obrázek 14) lze vyčíst, že u použité kombinace Fibonacciho kódování pro pozici a přímého kódování pro délku došlo ke zhoršení efektivity ve srovnání s předchozími dvěma testy. Například vzhledem k prvnímu testu v kapitole 7.1.1 došlo k navýšení kompresního poměru o 5,95% (porovnáno pro nejlepší výsledky algoritmu optimal LZSS obou testů). Tvar grafu a volbu nejlepší maximální délky shody, což je 16B, i zde výrazně ovlivňuje přímé kódování parametru délka. Nejlepší velikost okénka nadále zůstává 65 536B (64kB), tedy ta největší z předem zvolených.

Rozdíl mezi efektivitou optimal LZSS a original LZSS slabě narostl, jinak nedošlo k žádným výrazným změnám mezi jednotlivými algoritmy.

7.1.4 Fibonacciho kódování pozice i délky



Obrázek 15 – Graf alice29.txt – Fibonacciho kódování (pozice, délka)

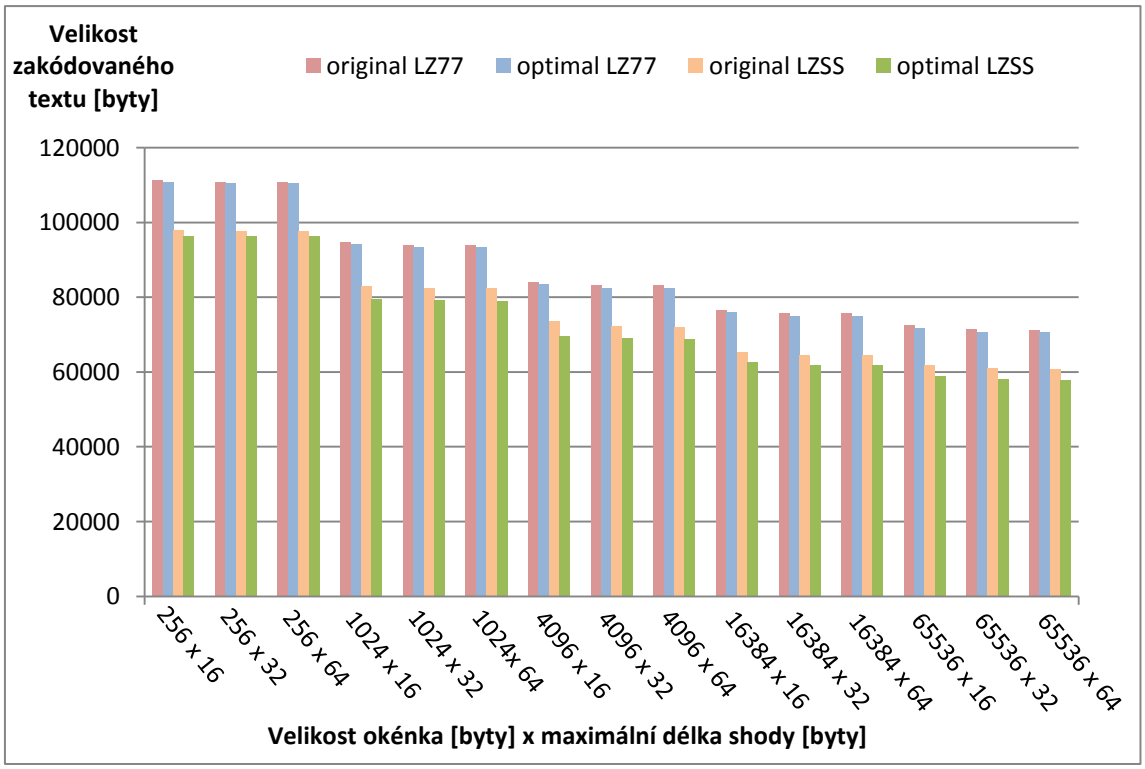
Z výsledků testu, zaznamenaných v grafu na obrázku 15, je zřejmé, že se efektivita všech algoritmů používajících Fibonacciho kódování pro pozici i délku v celkovém měřítku mírně zvýšila vůči testu z kapitoly 7.1.3. Avšak pro nejlepší výsledky algoritmu optimal LZSS obou testů se kompresní poměr zvýšil o 1,41%. Hodnoty tohoto testu jsou stále značně vysoké i oproti výsledkům testu z kapitoly 7.1.1, a to o 7,36% (porovnáno pro nejlepší výsledky algoritmu optimal LZSS obou testů). Znovu jako nejvhodnější varianta velikosti okénka a maximální délky shody vychází 65 536B (64kB) a 64B.

7.1.5 Přímé kódování pozice, B-blokové kódování délky

U B-blokového kódování je velikost zakódovaného textu závislá na zvoleném parametru *b*, který představuje základ kódování. Jeho výběr závisí na maximální kódované hodnotě a na četnosti jednotlivých hodnot. Vzhledem k implementaci tohoto kodéru je vhodné volit parametr *b* v mocninách 2 pro větší čísla. Maximální délka shody může nabývat hodnot 16B, 32B, 64B, proto pro testování parametru *b* zvolím menší hodnoty: 2, 3, 4, 5, 6. Výsledky viz následující tabulka 2. Velikost okénka je pro tento test konstantní – 256B a testovací algoritmus je optimal LZSS.

Parametr *b*	**16B** V [byte]	**32B** V [byte]	**64B** V [byte]
2	97033	96846	96821
3	97870	97666	97640
4	**96482**	**96280**	**96254**
5	100394	100172	100144
6	100097	99877	99849

Tabulka 2 – B-blokové kódování – ideální parametr b pro kódování délky (V - velikost zakódovaného textu v bytech, nejlepší výsledky jsou zvýrazněny tučně)



Obrázek 16 – Graf alice29.txt – přímé kódování (pozice), B-blokového kódovaní (délka), $b = 4$

Na hodnotách grafu (obrázek 16) můžeme pozorovat nejefektivnější kombinaci kódování – přímé kódování pro pozici a B-blokové kódování pro délku, kde ideální parametr *b* byl nastaven na hodnotu 4. Opět jako nejvýhodnější velikost okénka a maximální délka shody se ukázala varianta 65 536B (64kB) a 64B. Oproti doposud nejlepšímu testu v kapitole 7.1.1 tato kombinace kódování zlepšila kompresní poměr o 0,03% tedy o 48B (porovnáno pro nejlepší výsledky algoritmu optimal LZSS obou testů).

7.1.6 Shrnutí výsledků

Všechny testy dopadly podle očekávání a výsledky těch nejefektivnějších variant jsou shrnuty v tabulce 3, včetně kompresních poměrů.

Mimo jiné, chybějící použití B-blokového kódování pro parametr pozice nebylo uváděno z důvodů nenalezení ideálního parametru *b*. Pro takový rozsah hodnot, jakých může nabývat pozice pro jednotlivé velikosti okénka, není možné použít pouze jeden parametr *b*.

Pro varianty kódování jsem použila následující zkratky:

PP – přímé kódování pozice i délky pro velikost okénka 65 536B a maximální délku shody 16B

PF – přímé kódování pozice, Fibonacciho kódování délky pro velikost okénka 65 536B a maximální délku shody 64B

PB – přímé kódování pozice, B-blokové kódování délky pro velikost okénka 65 536B, maximální délku shody 64B a parametr $b = 4$

Kódování	**Original LZ77**		**Optimal LZ77**		**Original LZSS**		**Optimal LZSS**	
	V[byte]	KP[%]	V[byte]	KP[%]	V[byte]	KP[%]	V[byte]	KP[%]
PP	71932	47,30	71932	47,30	61046	40,14	57950	38,10
PF	73880	48,58	72990	47,99	63712	41,89	60254	39,62
PB	**71291**	**46,87**	**70664**	**46,46**	**60885**	**40,03**	**57902**	**38,07**

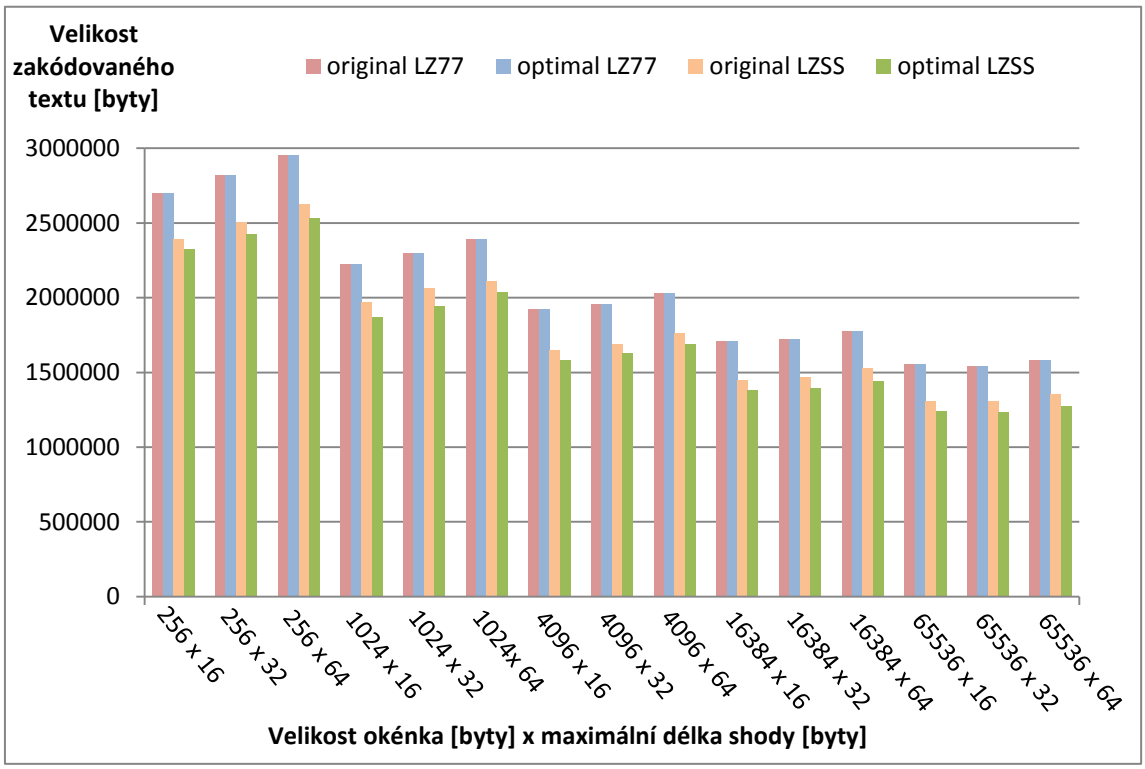
Tabulka 3 – Shrnutí nejlepších výsledků alice29.txt (V - velikost zakódovaného textu v bytech, KP - kompresní poměr v %, nejlepší výsledky jsou zvýrazněny tučně)

7.2 Testování souborů bible.txt

Také soubor bible.txt, který je velmi odlišný od dříve testovaného souboru alice29.txt (7.1), bude podrobně testován. Hlavním účelem je ověření výsledků předešlých testů. Pro soubor bible.txt jsou použity stejné hodnoty, velikost okénka 256B, 1kB, 4kB, 16kB, 64kB, maximální délka shody 16B, 32B, 64B a stejné kombinace kódování. Pro B-blokové kódování budu znovu zjišťovat ideální hodnotu parametru *b*.

Vzhledem k povaze následujících testů a jejich očekávanou podobností s testy předchozími již nemá význam detailně popisovat jednotlivě vytvořené grafy, pouze případné odlišnosti.

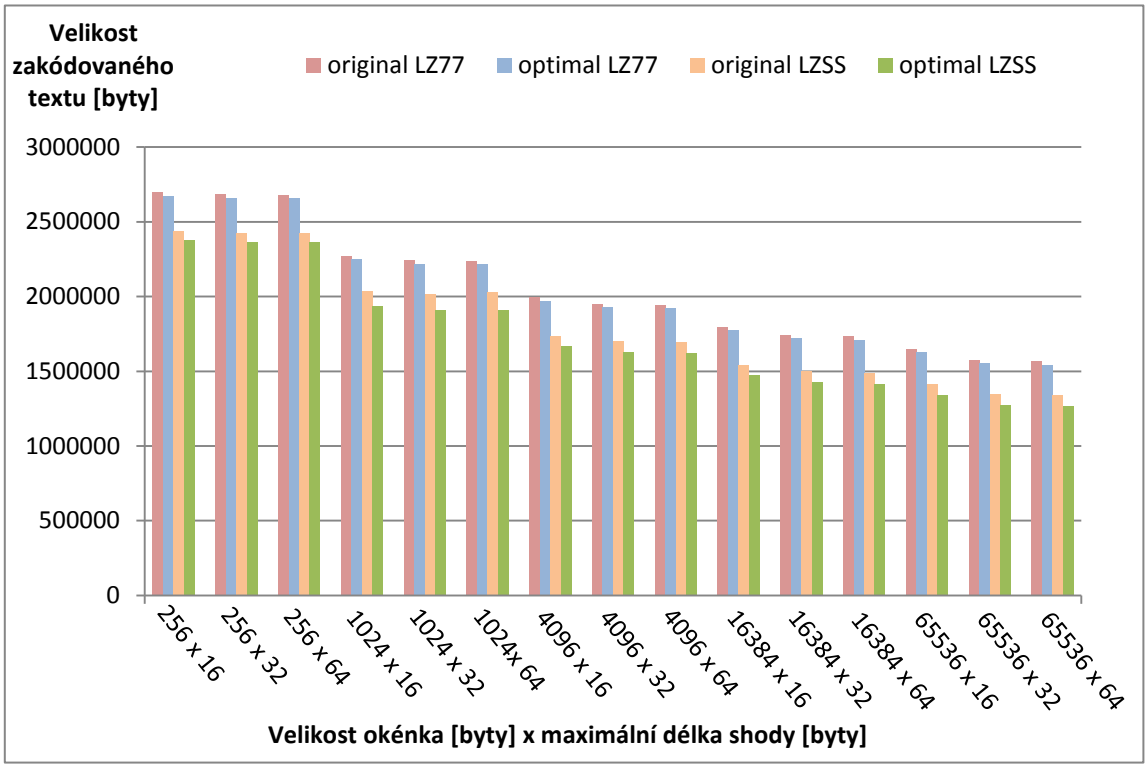
7.2.1 Přímé kódování pozice i délky



Obrázek 17 – Graf bible.txt – přímé kódování (pozice, délka)

Graf (obrázek 17) přímého kódování pozice i délky testovacího souboru bible.txt je svým tvarem totožný s grafem přímého kódování pozice i délky testovacího souboru alice29.txt (7.1.1). I zde se jako nejlepší výstupní kódování ukázalo optimal LZSS pro velikost okénka 65 536B (64kB) a maximální délka shody 16B.

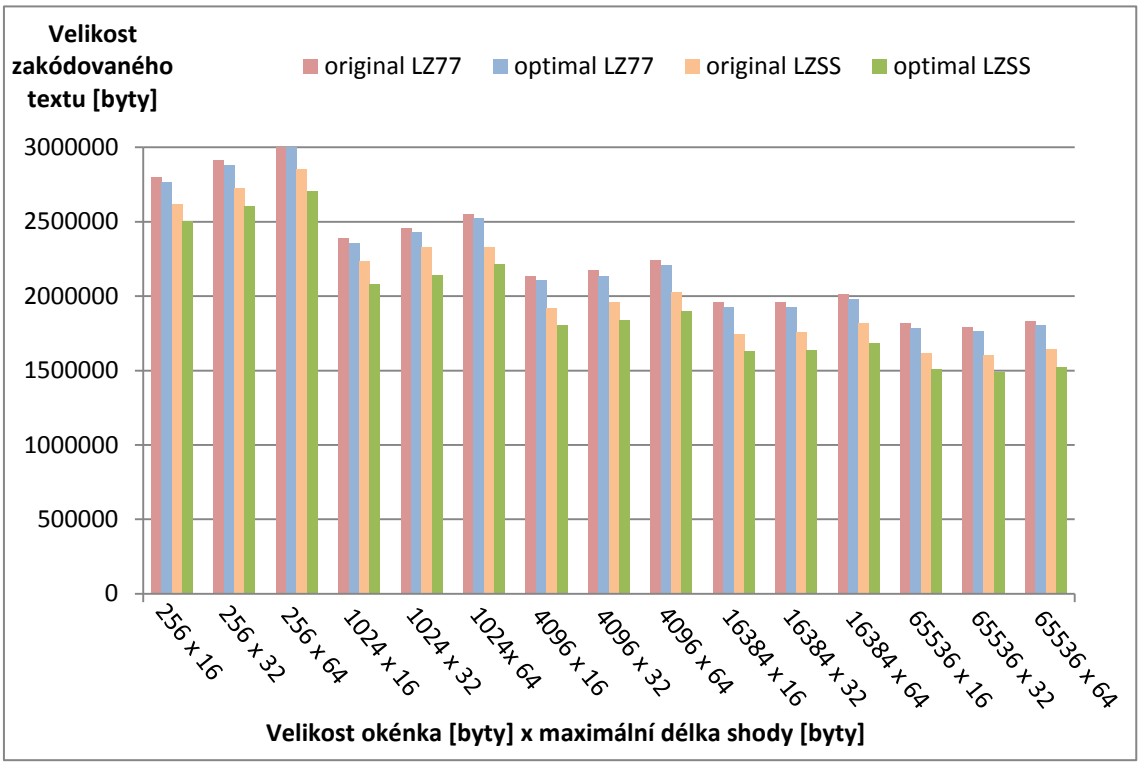
7.2.2 Přímé kódování pozice, Fibonacciho kódování délky



Obrázek 18 – Graf bible.txt – přímé kódování (pozice), Fibonacciho kódování (délka)

Opět tvar tohoto grafu, znázorněného na obrázku 18, není nijak odlišný od grafu z kapitoly 7.1.2. Nejlepší výsledek byl dosažen pro velikost okénka 65 536B (64kB) a maximální délku shody 64B u optimal LZSS.

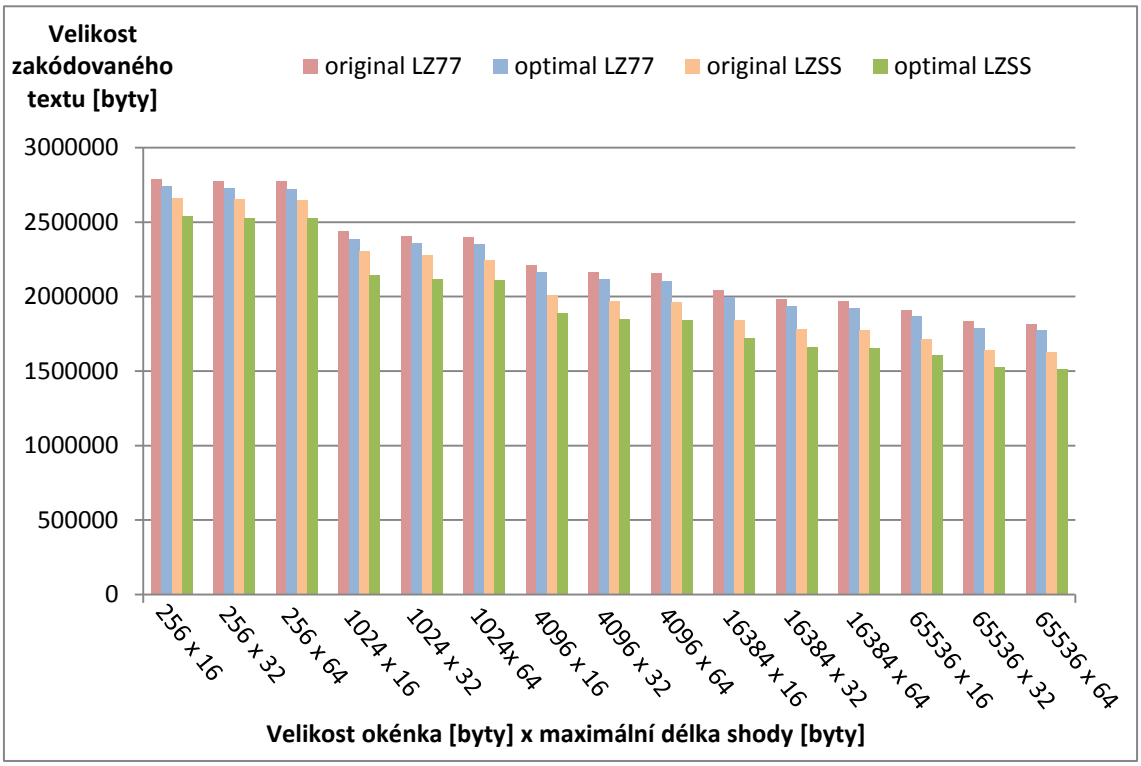
7.2.3 Fibonacciho kódování pozice, přímé kódování délky



Obrázek 19 – Graf bible.txt – Fibonacciho kódování (pozice), přímé kódování (délka)

U grafu z obrázku 19 dochází oproti grafu z kapitoly 7.1.3 k mírné odlišnosti. Jako nejefektivnější byla vyhodnocena tato kombinace - velikost okénka 65 536B (64kB) a maximální délka shody 32B. Jedná se však pouze o velmi malou odchylku. Je ověřeno, že celkově tato varianta výstupních kódování není příliš efektivní.

7.2.4 Fibonacciho kódování pozice i délky



Obrázek 20 – Graf bible.txt – Fibonacciho kódování (pozice, délka)

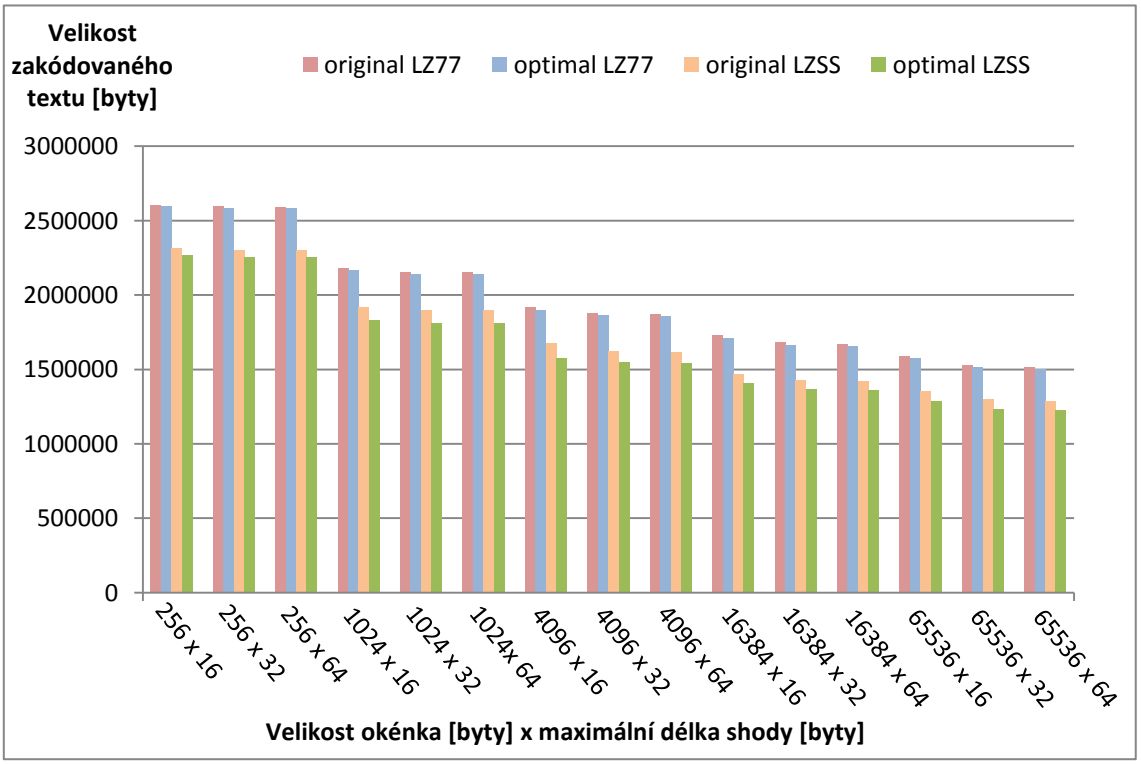
I zde graf (obrázek 20) potvrzuje správnost výsledků předchozího testu Fibonacciho kódování pozice i délky souboru alice29.txt (7.1.4). Znovu při velikosti okénka 65 536B (64kB) a maximální délce shody 64B vyšel nejlepší výsledek pro optimal LZSS.

7.2.5 Přímé kódování pozice, B-blokové kódování délky

Opět otestuji efektivitu B-blokového kódování pro menší hodnoty: 2, 3, 4, 5, 6, kterých bude parametr *b* nabývat. Maximální délka shody může nabývat hodnot 16B, 32B, 64B. Výsledky viz následující tabulka 4. Velikost okénka je pro tento test konstantní – 256B a testovací algoritmus je optimal LZSS.

Parametr *b*	**16B** V [byte]	**32B** V [byte]	**64B** V [byte]
2	2288488	2279428	2278434
3	2303875	2294141	2293078
4	**2264757**	**2255072**	**2254007**
5	2355761	2345213	2344056
6	2345008	2334624	2333466

Tabulka 4 – B-blokové kódování – ideální parametr b pro kódování délky (V - velikost zakódovaného textu v bytech, nejlepší výsledky jsou zvýrazněny tučně)



Obrázek 21 – Graf bible.txt – přímé kódování (pozice), B-blokového kódovaní (délka), $b = 4$

Nejlepších výsledků dosáhl parametr $b = 4$. Opětovně byla vítězem metoda optimal LZSS pro velikost okénka 65 536B (64kB) a maximální délku shody 64B. Celkově pak byla

tato varianta přímého pozice a B-blokového kódování délky nejefektivnější, shrnuto v následující kapitole 7.2.6.

7.2.6 Shrnutí výsledků

Ověření výsledků testů prováděných na souborech alice.txt a bible.txt jednoznačně prokázalo, že nejefektivnější kompresní algoritmus je navržený optimal LZSS, dále, podle očekávání totožné, tři nejefektivnější varianty kódování jsou shrnuty v tabulce 5, včetně kompresních poměrů.

B-blokového kódování pro parametr pozice nebylo použito ze stejného důvodu jako u souboru alice29.txt.

Pro varianty kódování jsem použila následující zkratky:

PP – přímé kódování pozice i délky pro velikost okénka 65 536B a maximální délku shody 16B

PF – přímé kódování pozice, Fibonacciho kódování délky pro velikost okénka 65 536B a maximální délku shody 64B

PB – přímé kódování pozice, B-blokové kódování délky pro velikost okénka 65 536B, maximální délku shody 64B a parametr $b = 4$

Kódování	**Original LZ77**		**Optimal LZ77**		**Original LZSS**		**Optimal LZSS**	
	V[byte]	KP[%]	V[byte]	KP[%]	V[byte]	KP[%]	V[byte]	KP[%]
PP	1552513	38,36	1552513	38,36	1308312	32,32	1244607	30,75
PF	1566078	38,69	1544940	38,17	1342801	33,18	1266705	31,30
PB	**1516588**	**37,47**	**1501605**	**37,10**	**1288048**	**31,82**	**1222591**	**30,21**

Tabulka 5 – Shrnutí nejlepších výsledků bible.txt (V - velikost zakódovaného textu v bytech, KP - kompresní poměr v %, nejlepší výsledky jsou zvýrazněny tučně)

7.3 Testování dalších souborů

Po podrobném testování dvou odlišných souborů alice29.txt a bible.txt získané znalosti využiji při testování dalších tří souborů. Soubory aaa.txt, lcet10.txt a world192.txt budou testovány pouze pro následující tři nejlepší varianty kódování z předešlých testů (uvedeno včetně používaných zkratek):

PP – přímé kódování pozice i délky pro velikost okénka 65 536B a maximální délku shody 16B

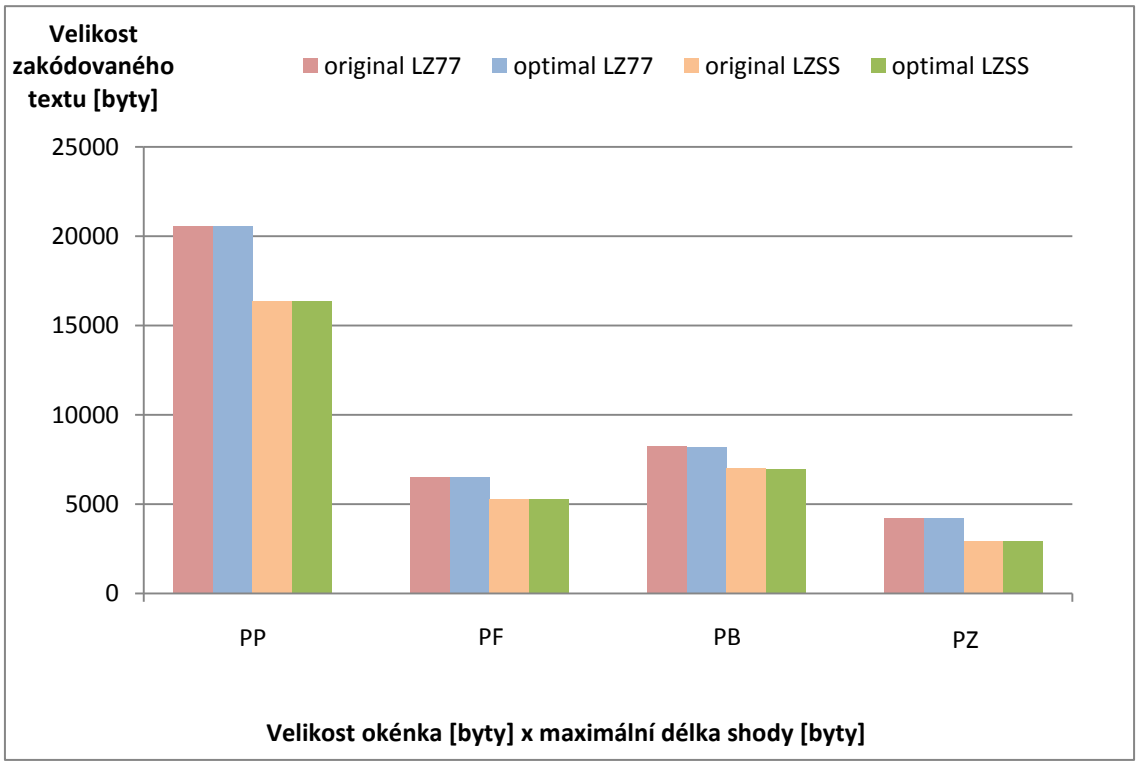
PF – přímé kódování pozice, Fibonacciho kódování délky pro velikost okénka 65 536B a maximální délku shody 64B

PB – přímé kódování pozice, B-blokové kódování délky pro velikost okénka 65 536B, maximální délku shody 64B a parametr $b = 4$

7.3.1 Soubor aaa.txt

Vzhledem k povaze souboru jsem předpokládala, že již otestovaná nastavení nemusí být těmi nejlepšími. Po vlastní úvaze jsem přidala další variantu kódování (uvedeno včetně používané zkratky):

PZ – přímé kódování pozice i délky pro velikost okénka 256B a maximální délku shody 64B



Obrázek 22 – Graf aaa.txt - testování

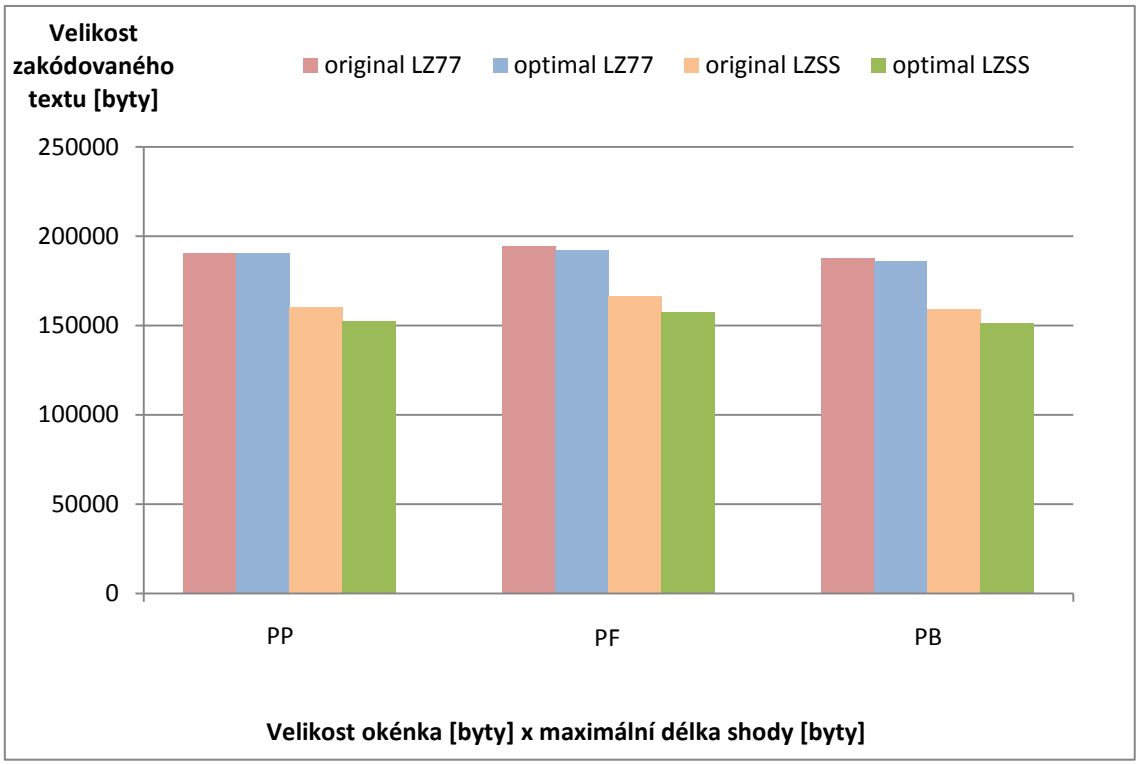
Kódování	Original LZ77		Optimal LZ77		Original LZSS		Optimal LZSS	
	V[byte]	KP[%]	V[byte]	KP[%]	V[byte]	KP[%]	V[byte]	KP[%]
PP	20594	20,59	20594	20,59	16408	16,41	16408	16,41
PF	6544	6,54	6544	6,54	5276	5,28	5276	5,28
PB	8275	8,28	8208	8,21	7034	7,03	6947	6,95
PZ	**4235**	**4,24**	**4235**	**4,24**	**2932**	**2,93**	**2932**	**2,93**

Tabulka 6 – Shrnutí výsledků aaa.txt (V - velikost zakódovaného textu v bytech, KP - kompresní poměr v %, nejlepší výsledky jsou zvýrazněny tučně)

Má úvaha se ukázala jako správná. Jelikož soubor aaa.txt obsahuje 100 000 znaků „a“, velikost okénka je limitována maximální délkou shody, tedy v našem případě nejlepší variantou se stalo přímé kódování pozice i délky pro velikost okénka 265B a maximální délku shody 64B.

Optimalizované algoritmy optimal LZ77 a optimal LZSS zde v porovnání s původními nepřináší žádné zvýšení efektivity. Celkově však efektivita výrazně vzrostla, ale s takovými daty se v reálném světě často nesetkáme.

7.3.2 Soubor lcet10.txt



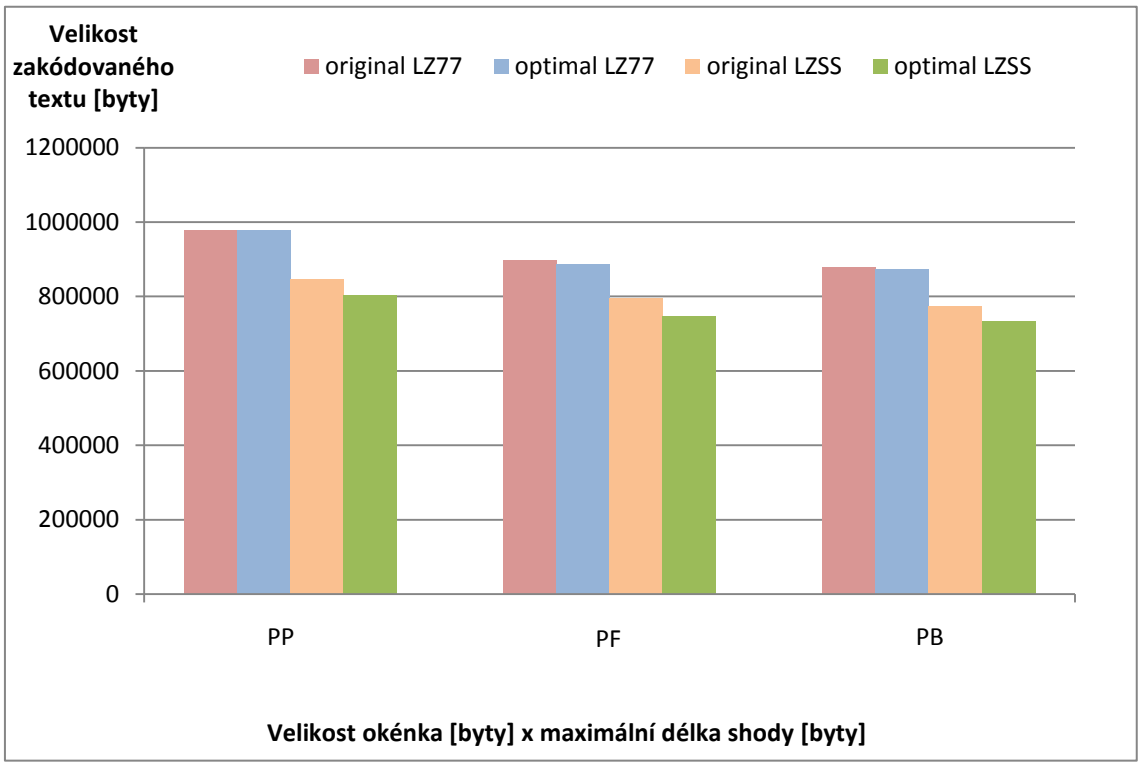
Obrázek 23 – Graf lcet10.txt - testování

Kódování	Original LZ77		Optimal LZ77		Original LZSS		Optimal LZSS	
	V[byte]	KP[%]	V[byte]	KP[%]	V[byte]	KP[%]	V[byte]	KP[%]
PP	190824	44,72	190824	44,72	160352	37,57	152631	35,77
PF	194600	45,60	192161	45,03	166276	38,96	157471	36,90
PB	**187992**	**44,05**	**186238**	**43,64**	**159087**	**37,28**	**151505**	**35,50**

Tabulka 7 – Shrnutí výsledků lcet10.txt (V - velikost zakódovaného textu v bytech, KP - kompresní poměr v %, nejlepší výsledky jsou zvýrazněny tučně)

Dle očekávání se výsledky efektivity jednotlivých variant shodují s testy souborů alice.29.txt a bible.txt. Nejefektivnější výstupní kódování je optimal LZSS pro přímé kódování pozice a B-blokové kódování délky, kde velikost okénka je 65 536B (64kB) a maximální délka shody je 64B.

7.3.3 Soubor world192.txt



Obrázek 24 – Graf world192.txt - testování

Kódování	Original LZ77		Optimal LZ77		Original LZSS		Optimal LZSS	
	V[byte]	KP[%]	V[byte]	KP[%]	V[byte]	KP[%]	V[byte]	KP[%]
PP	976955	39,50	976955	39,50	846117	34,21	803800	32,50
PF	897593	36,29	886769	35,85	796416	32,20	747486	30,22
PB	**878436**	**35,52**	**871329**	**35,23**	**774624**	**31,32**	**732771**	**29,63**

Tabulka 8 – Shrnutí výsledků world192.txt (V - velikost zakódovaného textu v bytech, KP - kompresní poměr v %, nejlepší výsledky jsou zvýrazněny tučně)

Opět se výsledky efektivity jednotlivých variant s drobnými rozdíly shodují s předchozími testy. Nejefektivnější kódování je znovu optimal LZSS pro přímé kódování pozice a B-blokové kódování délky, kde velikost okénka je 65 536B (64kB) a maximální délka shody je 64B.

8 Závěr

V první teoretické části této práce jsem se seznámila s pojmem obecné komprese a jejího využití, základním i více sofistikovaným dělením kompresních metod. Detailně jsem popsala metodu LZ77, metodu LZSS, princip posuvného okénka a jeho vlastnosti, postup při kompresi a zpětné dekompresi obou těchto algoritmů. Uvedla jsem i několik příkladů pro lepší znázornění problematiky. Dále jsem vysvětlila princip opožděného vyhodnocování, u něhož jsem také uvedla názorné příklady.

Ve druhé praktické části jsem realizovala rozbor a nástin možného řešení problému s nejlepším kódováním. Veškeré získané poznatky jsem použila při tvorbě programu, který zvyšuje efektivitu obou kompresních metod LZ77 a LZSS především pomocí opožděného vyhodnocování a zpětného průchodu sekvence. Následně jsem provedla sérii testů na pětici souborů z Canterbury korpusu.

Na souborech alice29.txt a bible.txt jsem detailně testovala původní kodéry original LZ77 a original LZSS i nově vytvořené optimal LZ77 a optimal LZSS pro kombinace různých typů kódování čísel, velikosti okénka a maximální délky shody. Na vzniklých grafech je velmi dobře zobrazeno, jak výrazně ovlivňuje výsledek použité kódování pro pozici a délku. U souboru alice29.txt došlo v nejlepším případě ke zlepšení kompresního poměru na 38,07%, u souboru bible.txt až na 30,21%. Podle očekávání byl optimal LZSS celkově nejefektivnější algoritmus ze čtveřice testovaných.

Na základě výsledků těchto podrobných testů jsem vybrala tři nejefektivnější kombinace předchozích parametrů a ty následně použila na zbylé soubory – přímé kódování pozice i délky pro velikost okénka 65 536 bytů a maximální délky shody 16 bytů; přímé kódování pozice, Fibonacciho a B-blokové kódování délky pro velikost okénka 65 536 bytů a maximální délky shody 64 bytů. U B-blokového kódování byl po několika testech zvolen nejvhodnější parametr $b = 4$. Výsledné grafy potvrdili správnost zvolených kodérů a dalších veličin. U souboru lcet10.txt došlo v nejlepším případě ke zlepšení kompresního na 35,5%, u souboru world192.txt dokonce na 29,63%. Soubor aaa.txt svou povahou ovlivnil výsledky testování, kdy došlo ke zlepšení kompresního poměru na vynikajících 2,93%, a proto jim nelze přikládat přílišnou váhu, jelikož v reálném světě se s podobnými daty setkáváme jen málo nebo vůbec.

V budoucnu by bylo vhodné řešit použití ideálního kódování znaků, namísto pevného počtu bitů pro jeden znak, což by mělo ve výsledku pozitivně ovlivnit velikost zakódovaného textu.

9 Literatura

[1] SALOMON, D. *Data Compression: The Complete Reference*. 4th ed. London: Springer, 2007. ISBN 978-1-84628-602-5.

[2] PLATOŠ, J. *Slovní metody komprese textu.* Ostrava: Vysoká škola báňská – Technická univerzita Ostrava. Fakulta elektrotechniky a informatiky. Katedra informatiky, 2006. Vedoucí diplomové práce doc. Mgr. Jiří Dvorský, Ph.D.

[3] VEČERKA A. *Komprese dat* [online]. © 2006-2008, poslední aktualizace 25.6.2009 [cit. 2012-04-01]. URL <http://phoenix.inf.upol.cz/esf/ucebni/komprese.pdf>.

[4] SAYOOD, K. *Introduction to Data Compression.* 3rd ed. San Francisco: Morgan Kaufmann, 2006. ISBN 0-12-620862-X.

[5] ČAPEK, J., FABIÁN, P. *Komprimace dat: principy a praxe.* 1. vyd. Praha: Computer Press, 2000. ISBN 80-7226-231-9.

[6] HORDĚJČUK, V. *Osobní stránky* [online]. © 2012, poslední aktualizace 21.3.2012 [cit. 2012-04-01]. URL < http://voho.cz/wiki/informatika/kodovani/komprese/>.

[7] BELL, T. *The Canterbury Corpus* [online]. © 2001, poslední aktualizace 8.1.2001 [cit. 2012-04-01]. URL < http://corpus.canterbury.ac.nz/>.

Seznam příloh

Přiložené medium CD-ROM obsahuje dva adresáře:

I. **software** – program OptimalLZ (implementováno v jazyce C#), včetně testovacích souborů
II. **text** – vlastní text bakalářské práce ve formátu PDF/A